Multilayer Perceptron Neural Network for Detection of Encrypted VPN Network Traffic

Multilayer Perceptron Neural Network for Detection of Encrypted VPN Network Traffic

Shane Miller
Faculty of Computing and Engineering
Ulster University
Northern Ireland
miller-s5@ulster.ac.uk

Kevin Curran
Faculty of Computing and Engineering
Ulster University
Northern Ireland
Kj.curran@ulster.ac.uk

Tom Lunney
Faculty of Computing and Engineering
Ulster University
Northern Ireland
Tf.lunney@ulster.ac.uk

Abstract— **There has been a growth in popularity of privacy in the personal computing space and this has influenced the IT industry. There is more demand for websites to use more secure and privacy focused technologies such as HTTPS and TLS. This has had a knock-on effect of increasing the popularity of Virtual Private Networks (VPNs). There are now more VPN offerings than ever before and some are exceptionally simple to setup. Unfortunately, this ease of use means that businesses will have a need to be able to classify whether an incoming connection to their network is from an original IP address or if it is being proxied through a VPN. A method to classify an incoming connection is to make use of machine learning to learn the general patterns of VPN and non-VPN traffic in order to build a model capable of distinguishing between the two in real time. This paper outlines a framework built on a multilayer perceptron neural network model capable of achieving this goal.**

Keywords—classification; flow statistics; neural networks; VPN; Networking

I. INTRODUCTION

Virtual Private Networks (VPNs) are becoming a popular method for hackers to hide their online activities [1]. This is helped along by the increase in ease of use of VPNs which are no longer just a tool for remotely accessing enterprise resources. If bad actors wish to remotely access an enterprise network in order to steal company and trade secrets, they can use a VPN (or multiple VPNs) in order to hide their own location or to make it appear as if someone else was infiltrating the network [2]. There have been a few notable cases of this happening in recent years, such as the Sony Pictures incident from 2014 where confidential data including personal information about employees was stolen [3]. Other attacks of note are the various data breaches which have been occurring for the last number of years, such as the LinkedIn breach of 2012 which was only discovered in 2016 [4]. Approximately 167 million account details including emails and passwords were stolen.

If the criminals that carried out these attacks were making use of VPNs to hide their identities, then tracking them down can prove to be challenging if not impossible. Knowing whether a VPN has been used or not could help in the tracking down of those responsible for attacks such as those mentioned above. Traffic classification techniques such as data analytics and machine learning algorithms can be employed to solve this problem.

We present here a flow statistic-based classification method using a multi-layered perceptron neural network to distinguish between TCP layer web traffic that is being transmitted over an encrypted OpenVPN channel and normal traffic that may be encrypted or not. In this model, TCP flows captured from encrypted VPN sessions are analysed alongside other TCP flows that are captured from a mix of general browsing sessions. Flow statistics including time-based statistics and other metrics are extracted from the captured traffic and compiled into a dataset. This dataset is analysed using Pearson's Correlation Coefficient algorithm to determine the strongest features in classifying whether the traffic belongs to a VPN or not. The aim of the model is to identify whether an incoming connection to a web server originated from a VPN server. The purpose of the limited scope is to identify a working model which can then be extended.

The remainder of this paper is organised as follows: Section 2 presents an overview of related work in the area. Section 3 describes the capture of the flow-based dataset and the setup of the VPN being used to generate the traffic. Section 4 describes the experiments run in Weka using the dataset and Section 5 presents the results from the experiments. Section 6 presents conclusions and possible future work.

II. BACKGROUND

Research involving the use of flow statistics as a classification metric began in the early 1990s [5], [6]. Statistical features such as packet length, inter-arrival times and the duration of the flow were used to analyse 3 million TCP connections that took place during 15 wide-area network traffic traces and derive the protocols contained in the traces. Subsequent research uses statistics gained from the first few packets on both directions of a flow to gain efficiency and increase the possibility of traffic classification in real time [7], [8]. To further increase the efficiency of classification in a highly scalable and high-speed network, signature-based traffic identification schemes were proposed. The goal of these schemes were to provide an increase in efficiency without compromising classification accuracy by combining the advantages of accuracy found in deep packet analysis (DPI) and the advantages of speed found in port-based classification approaches [9], [10].

Research into classification of network traffic using flow-based statistics is addressed in the literature with an apparent focus on detecting specific types of traffic [11]. One proposed model attempts to characterise Peer to Peer (P2P) traffic using features extracted from multiple captured flows which are then clustered in order to extract P2P application behaviour [12]. Another proposed method attempts to provide an accurate model for identifying and detecting malicious botnet traffic using supervised machine learning along-side flow-based features [13]. Attempts to detect and classify encrypted traffic include a proposed method where encrypted network traffic will be classified by calculating entropy-based features from a packet payload and using that to create a supervised machine learning model [14]. Another proposed framework is a DPI system that inspects encrypted network traffic without actually decrypting it in an effort to maintain the privacy of the communication [15]. Traffic classification with a focus on detecting traffic that is encapsulated within a Proxied connection or encrypted VPN is challenging and only now becoming a field of study. One proposed method attempts to deal with the problem of end users using proxies as a means to hide their identity by creating a data driven machine learning based approach using a mixture of log-files to represent realistic proxy use [16].

III. Dataset and VPN Setup

A. Dataset Capture

To create a representative dataset of VPN and non-VPN values, real network data was captured using a number of tools. Wireshark formed the basis of the packet capture for this dataset. The computer system used to capture the traffic was an Ubuntu 16.04 based virtual machine running on a Windows 10 host. The network connection used in the experiment is a virtualised Intel PRO gigabit ethernet card. While Linux might not be a mainstream operating system for the standard user, it is popular among "bad actors" because of its ability to be customised heavily according to the user's preferences. It allows for a finer degree of control over some of the internal systems included such as networking stack. Using built in tools, it is easy to automate connections and disconnections to different networks and different network interfaces. This was a particularly helpful feature when dealing with the capture of VPN based packets. In normal operation, a connection to a VPN starts with a typical TCP "hello" sequence and key exchange. Once the connection is setup, it is only taken down whenever the user stops using the VPN. The connection is a TCP connection between the user's machine and the VPN server that is maintained until the user closes the connection. This created some problems with the NetMate flow statistic calculation tool as it would only output one set of statistics for a capture session of ten hours. This was deemed to be an unrealistic scenario. This problem with NetMate was solved using a Linux bash shell script and the Linux system's automatic task scheduling tool; *cron*. The shell script contains a one-line command that initiates a connection to the VPN and sets a timeout value of 590 seconds. The 'timeout' causes the command to quit after the set amount of time and 'openvpn' is the command that will be affected by 'timeout'. Openvpn is a type of VPN server that can be installed easily on many systems and is the VPN used to generate the VPN packets needed for the dataset. In this command 'openvpn' initiates the connection to the Openvpn server that is described in the config file i.e. *<openvpn-config-file>.ovpn*. Combining this command with *"timeout 590s 'openvpn-config-file'.ovpn"* the automatic scheduling tool *cron* is straightforward. The goal was to have this command run every ten minutes, so the timeout value is set to 590 seconds which leaves enough time for the connection to completely close. Then, 10 seconds later the command is run once again. This repeats for as long as the command is listed in *cron*. To instruct *cron* to run this command as required, the file */etc/crontab* is edited with administrative privileges. A line is added to the file detailing the command to be run and how often it should be run. The line added to run the connection shell script every 10 minutes is:
"/10 * * * * root sh /path-to-file/connect.sh"*. From left to right, the headings of the crontab file stand for: minutes, hour, day of month, month, week of month, user to run command under and the actual command. This will instruct *cron* to run the shell script every 10 minutes of every day of every month as the user 'root'. With the connection to the VPN automated and refreshing every ten minutes, the next task was to generate the network traffic to be captured by Wireshark. There are tools available to generate random packets based on specific attributes to simulate certain network environments. However, it was preferred if the data could be captured from realistic browsing practices. Due to the amount of data that would be required, it would be infeasible for a person to sit and visit websites for 24 hours a day, 7 days a week. A modified version of the automated browsing Python script from the proxy detection work was used in conjunction with a small selection of the most popular Alexa top 500 sites[1]. This ensures that the browsing involves some of the most recent and publicly available sites on the web as well as some of the most popular. For the script to browse sites in a relatively realistic fashion, the sleep method is used in conjunction with the *randint* method. The *sleep* function pauses the execution of the entire script. When combined with *randint*, it is possible to pseudo-randomly set the pause time for each occurrence of *sleep*. With the VPN connecting and disconnecting every 10 minutes, the *randint* method's minimum value was set to 10 seconds and the maximum value set to 300 seconds. This means that the website that is visited by the script will be displayed for a minimum of ten seconds and no longer than 5 minutes. In their paper, [17] showed that users judged web pages harshly in the first 10-30 seconds. After this time had passed it was likely that users would spend upwards of 2 minutes on the page. During the 10-minute VPN connection period, the browse script would visit a minimum of 2 web pages. The same method was used to generate the non-VPN samples with the only difference being that a VPN connection was not maintained during the browsing.

[1] https://www.alexa.com/topsites

B. NetMate

NetMate is a bidirectional flow exporter and analyser tool used to convert capture files of network traffic into flow records [18]. A TCP flow is a sequence of packets between two endpoints as defined by their source IP address and port to a destination IP address and port over a certain length of time [19]. A sequence like this will only be considered a flow if it is monitored in both directions. The packets captured from Wireshark meet this requirement so they are compatible with NetMate. The particular version of NetMate used for this dataset is Netmate-flowcalc which is a bundle comprising of NetMate v0.9.5 packaged with NetAI modules from v0.1 [20]. The output, if using one of the included rules files, takes the form of a comma separated list of values. Each column corresponds to an attribute or feature of the output. Figure *3* shows a list of the features generated by NetMate and a description of what they measure. The first five attributes generated are taken directly from the TCP packet header. They include the source IP address and port number; the destination IP address and port number and the protocol being used and these are omitted from the final list as they would cause the model to overfit. The rest of the attributes in the table are flow statistics that are calculated by *NetMate*. The statistics calculated for the majority of the remaining attributes are the minimum, mean, maximum and standard deviation. These features are similar to those produced by another project named "flowtbag".

The overall size of the dataset captured and processed through NetMate was 9829 flows with 3569 flows representing VPN traffic and 6260 flows representing Non-VPN traffic. These were labelled *vpn* and *normal*. This overall dataset was split into three separate sets; one for training, one for testing and one for validation of the trained model. The original set was split into 80% training and 20% testing as this was regarded as a generally popular split according to the literature. The resulting testing set was then split further following the same original split, 80/20, resulting in the final testing and validation datasets. The training dataset contained 7863 instances. The final testing dataset contained 1257 instances and the validation dataset contained 253 instances.

C. VPN Setup: Streisand on AWS

We used the AWS platform to host a virtual machine which acts as a VPN server. The server that was chosen was the *t2.micro* Elastic Compute 2 (EC2) instance which is one of the more basic types, but more than adequate to run a fully featured VPN server for a small number of clients. It contains one virtual CPU core and one gigabyte of RAM. The software used to setup the server to allow it to provide VPN functionality is called Streisand[2]. Streisand sets up a new remote server with the Ubuntu 16.04 operating system that is capable of running various services such as L2TP/IPsec, OpenVPN and other methods of tunnelling network traffic via VPN.

[2] https://github.com/StreisandEffect/streisand

Attribute Name	Attribute Description
total_fpackets	Totals packets in the forward direction.
total_fvolume	Total bytes in the forward direction.
total_bpackets	Total packets in the backward direction.
total_bvolume	Total bytes in the backward direction.
fpktl	The min, mean, max and standard deviation from the mean of packet sizes in the forward direction.
bpktl	The min, mean, max and standard deviation from the mean of packet sizes in the backward direction.
flat	The min, mean, max and standard deviation from the mean of time between two packets in the forward direction.
biat	The min, mean, max and standard deviation from the mean of time between two packets in the backward direction.
duration	Total duration of the flow in microseconds.
active	The min, mean, max and standard deviation from the mean of time that the flow was active before going idle.
idle	The min, mean, max and standard deviation from the mean of time that the flow was idle before going active.
sflow_fpackets	Average number of packets in a sub flow in the forward direction.
sflow_fbytes	Average number of bytes in a sub flow in the forward direction.
sflow_bpackets	Average number of packets in a sub flow in the backward direction.
sflow_bbytes	Average number of bytes in a sub flow in the backward direction.
fpsh_cnt	Number of PSH flags set in packets in the forward direction.
bpsh_cnt	Number of PSH flags set in packets in the backward direction.
furg_cnt	Number of URG flags set in packets in the forward direction.
burg_cnt	Number of URG flags set in packets in the backward direction.
total_fhlen	Total bytes used for headers in the forward direction.
total_bhlen	Total bytes used for headers in the backward direction.

Fig. 1. Description of netmate statistical features.

The setup is heavily automated, relying on an automation tool named Ansible that is typically used to provision and configure files and packages on remote servers. The only input required from the user is to choose a cloud provider, physical region for the server and the API information for the cloud platform that the user wishes to set the server up on. Once this information has been provided, the script begins the creation and initial setup process for the remote server, installing the required software and tools needed. Once the server has been fully set up, several files are created on the user's local computer which contain instructions on getting started. For the experiments run, the OpenVPN protocol is the type of VPN being tested. The reason for choosing OpenVPN is because it is well-known and popular due to its very simple configuration. Servers and clients are widely supported across many different platforms i.e. an OpenVPN server running on a Linux/Unix based host can be accessed by a Windows client and vice versa. There are also client applications available for mobile platforms such as Android and iOS, making it suitable for on-the-go VPN use.

IV. WEKA EXPERIMENT

The Weka (Waikato Environment for Knowledge Analysis) workbench is a collection of standard machine learning algorithms and data pre-processing tools. It is designed to allow researchers to quickly apply existing methods of machine learning to new datasets [21]. Weka includes methods for many types of machine learning problem including: regression, classification, clustering and attribute selection. Weka includes a few ways of setting up experiments: *Explorer, Knowledge Flow, Experimenter and Workbench*. *Explorer* is a graphical user interface which provides access to all of the facilities of Weka using menu selection and forms. *Knowledge flow* is an

interface that allows you to visualise and control the stream of data when using larger datasets. A drawback of *Explorer* is that datasets are loaded in their entirety to the computers RAM, meaning that datasets that need a larger amount of memory than the computer can provide will not be able to be used. *Knowledge flow* enables a researcher to specify a data stream by connecting components representing data sources, pre-processing tools, learning algorithms, evaluation methods and visualisation modules. If the filters and learning algorithms are capable of incremental learning, then the dataset will be loaded into memory in increments causing memory to be saved. *Experimenter* is designed to answer the question of which learning algorithms and parameters values work best for the given problem. This can be accomplished manually using *Explorer.* However, *Experimenter* allows the researcher to automate the process by making it easy to run different classifiers with different parameters on multiple datasets, collect the results and performance statistics and then analyse them to see what combination works best for the given problem. The last interface is called *Workbench.* It is a unified graphical interface which incorporates features from the other three into one application. It is highly configurable and allows for the creation of a highly tailored interface.

The method used for the experiments is based on *Explorer.* This is the most straight-forward interface to using Weka and can be used to load in datasets, run experiments and analyse the results. Weka's native data storage method is the ARFF format, however it provides methods to convert data to ARFF from spreadsheets and databases. It can also accept comma-separated value (CSV) files, ASCII MATLAB files, LIBSVM etc. and also provides methods to convert them to the ARFF format. CSV files that have had the ARFF attribute information added to them manually can also be accepted as long as Weka is able to interpret the ARFF headers correctly. The ARFF header helps Weka to identify what is an attribute, what is an instance of the data and what are the classes (if any). Once the data has been loaded into Weka, it can be modified using the *Pre-process* tab. Modifications include the ability to remove attributes, add instances manually and apply various filters to the entire dataset. Modified versions of the dataset can be saved to their own file, leaving the original dataset intact for future use or modification. One of the most useful modifications is the ability to split the original dataset into separate training, testing and validation sets using the provided filters. The next tab is *Classify* and it is here that the various machine learning algorithms are trained to perform classification or regression and evaluate the results. The dataset that is loaded into the *pre-process* tab is seamlessly transferred across to the *classify* tab. If not already done, the dataset can be split into training and testing here using the "Test options" selections or, if already completed, the test set can be specified. The type of classification or regression algorithm can be chosen here as well as from the large selection that Weka provides, including Linear Regression, Multilayer Perceptron, Naïve Bayes and C4.5 decision tree algorithms. All of the classifiers are adjustable via another Weka dialogue which enables customisation via drop down menus and textboxes. Once setup the algorithm can be trained and tested by pressing the start button and once the model has been successfully trained and tested, the results are shown in the "Classifier output". The *Cluster* and *Associate* tabs were not used during the experiments described in this paper. The *Select attributes* tab gives access to several methods for attribute selection. This involves an attribute evaluator and a searching method. Both are selected and configured in the same way that options are chosen and configured in the other tabs. Selection can be performed using either the full dataset or by using cross-validation. The full dataset option was used for this paper's experiments. This allows the researcher to perform feature selection using a number of different feature selection algorithms.

A. Feature Selection

The feature selection model used was the Weka model *CorrelationAttributeEval* which is a model that is based on Pearson's Correlation Coefficient model. This is a measure of the linear correlation between two variables. The output of the model is a value between +1 and -1, where +1 is total positive linear correlation, 0 is no linear correlation and -1 is total negative linear correlation. In Weka, the search method *Ranker* is required to run *CorrelationAttributeEval. Ranker* ranks attributes by their individual evaluations i.e. from highest positive linear correlation to lowest negative linear correlation. It provides options to set a threshold by which attributes can be discarded, with the default being that no attributes are discarded. Through trial and error, the threshold for the experiments run in this paper was set to 0.5. This threshold is the cut-off point for whether an attribute of the data is kept as a feature or discarded. The result of this selection was a reduction from 44 features to the 10 features that were calculated to have a linear correlation above 5, a list of which is shown in figure 4. The lowest correlation was 0.544 for *total_fvolume* and the highest was 0.742 for *duration.*

TABLE I. CORRELATION COEFFICIENTS

Attribute Name	Ranking
total_fpackets	0.561
total_fvolume	0.544
max_fpktl	0.644
max_bpktl	0.724
duration	0.742
mean_active	0.677
max_active	0.57
std_active	0.55
fpsh_cnt	0.587
total_fhlen	0.561

B. Resampling the dataset

The original, full dataset was edited to create a training dataset of 7863 by resampling using the options outlined in figure 5. The 80/20 percent split can be seen in *sampleSizePercent* as "80".

To create the testing dataset, the original dataset was again resampled using the same criteria, but the *invertSelection* option was changed from false to true. This gives the opposite output of the first run and the output is a dataset of 1510 instances which is the 20% from the 80/20 split. These instances were then again resampled following the above steps to create the final testing dataset and the final validation dataset. Again, this followed the 80/20 percent split, with the 80% split being the testing set and the remaining the 20% being the validation set. The resulting datasets were 1257 instances for the testing dataset, or approximately 12% of the original dataset, and 253 instances for the validation dataset, approximately 2-3% of the original dataset.

C. Neural Network Setup

The Weka model used for classifying whether the instances from the dataset are traffic coming from a VPN or not is the *MultilayerPerceptron* model. This model is based on a standard artificial Neural Network that is trained using back propagation.

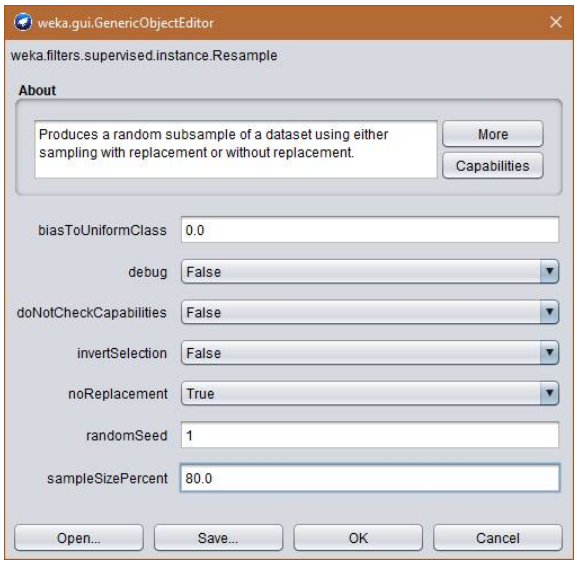

Fig. 2. The Weka Resample dialogue

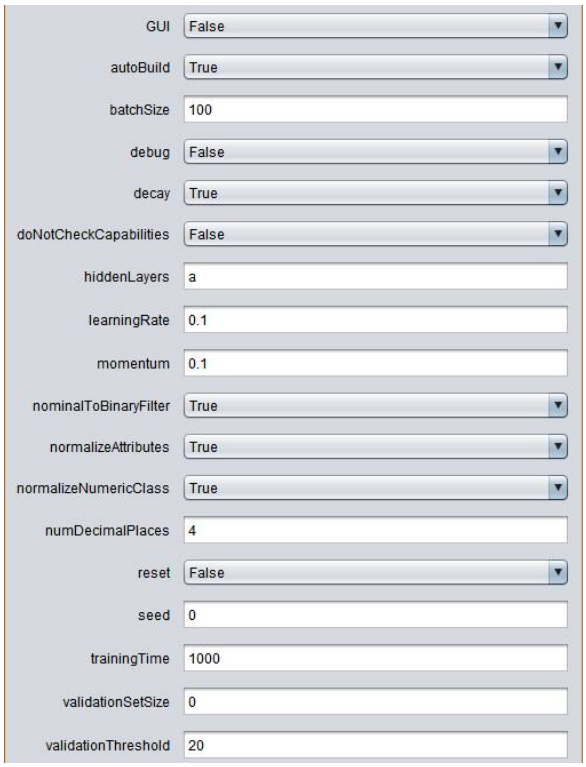

Fig. 3. Neural Network Weka Configuration

This model offers a large amount of customisation, with options to build a network by hand, let an algorithm build the network or a mixture of both. Figure 6 shows the configuration used to setup the neural network model used for the classification experiments. This setup was found to be the best performing configuration when compared to other networks with different configurations with regards to accuracy, training time and avoiding the problem of overfitting the data. Ideally for this model to be used in a real-world application, the training time

needs to be kept to a minimum whilst preserving as much accuracy as possible. For the purposes of classifying VPN and non-VPN traffic it was decided to allow Weka to create a fully connected network. This is accomplished by using the two options *autoBuild* and *hiddenLayers*. *Autobuild* is the option which instructs Weka whether to build a fully connected network or not with the two options being true or false. *HiddenLayers* is where the hidden nodes of the network are defined. The value shown in figure 6 is one of the provided wildcard values. 'a' creates a hidden layer by summing together the number of attributes and classes and then dividing the total in half. So, for 10 attributes and two classes, the number of hidden layers is set to six. Figure 7 shows the completed network, ready to be trained using the training dataset.

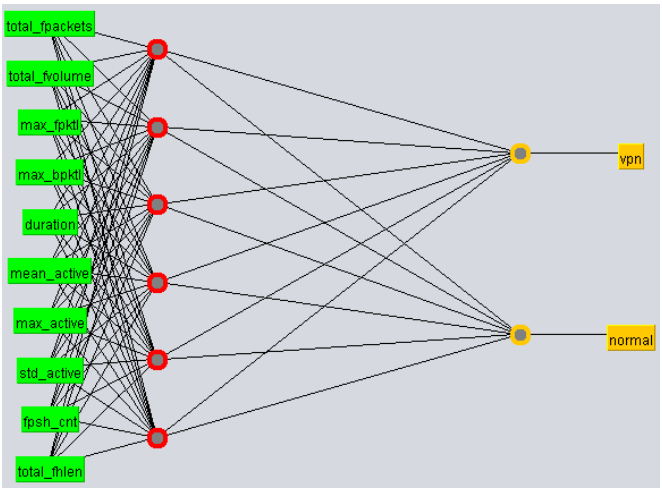

Fig. 4. Fully connected Neural Network

Once the model is configured, it is ready to train using the dataset currently loaded into the "Pre-process" tab. This network was trained using the above options and the total time taken to train the network and build the classification model was approximately 10 seconds using an 8-core processor. Testing was completed a few seconds later using the testing dataset. Validation of the result using the validation dataset was completed by loading it in as a test set and then re-evaluating the already trained model.

V. RESULTS

The results shown in Table 2 shows that the overall accuracy of detection for the neural network in the post-training test was approximately 94%. That is 1178 correctly classified instances out of a testing set of 1257. The average Precision, Recall and F-Measure of 0.937 backs up the accuracy and together they show that in the post training test using a separate sample of test data, the neural network is capable of distinguishing between VPN and non-VPN traffic when using flow-based features.

TABLE II VALIDATION TEST RESULTS

Correctly Classified Instances	1178 / 1257 (93.7152%)
Incorrectly Classified Instances	79 / 1257 (6.2848%)
True Positive Rate	0.895
False Positive Rate	0.039
Precision	0.929
Recall	0.895
F-Measure	0.912

Table 3 shows the confusion matrix for the post-training tests. It provides details on the correctly and incorrectly classified instances and how they are distributed as true positive and negative and false positive and negative. The goal is to keep the false positive and false negative as low as possible and table 3 shows that this has indeed been accomplished with the number of false positives (i.e. *Normal* traffic that has been incorrectly classified as a *VPN*) being 31 instances. The number of false negatives (i.e. *VPN* traffic that has been incorrectly classified as *Normal*) was 48 instances.

TABLE III CONFUSION MATRIX FOR VALIDATION TEST

Classified as	***VPN***	***Normal***
VPN	408	48
Normal	31	770

Table 4 shows the results for the validation test results. The validation test was performed on data that had been kept separate from the training process. The data was classified by the trained model as new data that it had never encountered before, therefore imitating real world conditions. The result was an accuracy rating of approximately 92% or 232 correctly classified instances out of 253.

TABLE IV TRAINING TEST RESULTS

Correctly Classified Instances	232 / 253 (91.6996%)
Incorrectly Classified Instances	21 / 253 (8.3004%)
Average True Positive Rate	0.848
Average False Positive Rate	0.043
Average Precision	0.918
Average Recall	0.848
Average F-Measure	0.881

Table 5 shows the confusion matrix for the validation test. This provides details on the correctly and incorrectly classified instances and how they are distributed as true positive and negative and false positive and negative. The number of false positives for the validation test was seven and the number of false negatives was 14.

TABLE V CONFUSION MATRIX FOR TRAINING TEST

Classified as	***VPN***	***Normal***
VPN	78	14
Normal	7	154

VI. CONCLUSION AND FUTURE WORK

In this paper we have successfully demonstrated the effectiveness of using a multi-layered perceptron neural network model trained using TCP flow-based features to classify incoming network traffic to a web server as either originating from an OpenVPN connection or not. The flow-based features discovered using Pearson's Correlation Coefficient model can be said to accurately distinguish between VPN and non-VPN traffic as shown by the resulting accuracies of 92% and 93%. Low false positive rates also make this an attractive approach for law enforcement uses. For future work we plan to investigate the usefulness of this model for other VPN protocols in an effort to build up the model's generalisation capabilities. We also plan to investigate other forms of network traffic other than web traffic to again improve the model's capabilities.

REFERENCES

[1] J. T. Harmening, "Virtual Private Networks," in *Computer and Information Security Handbook*, Third., Elsevier, 2017, pp. 843–856.

[2] S. Geetha and A. V. Phamila, *Combating Security Breaches and Criminal Activity in the Digital Sphere*. IGI Global, 2016.

[3] A. Peterson, "The Sony Pictures hack, explained.," *Washington Post*, 18-Dec-2014.

[4] T. Hunt, "Observations and thoughts on the LinkedIn data breach," *troyhunt.com*, 2016. [Online]. Available: https://www.troyhunt.com/observations-and-thoughts-on-the-linkedin-data-breach/. [Accessed: 06-Dec-2017].

[5] V. Paxson, S. Floyd, V. Paxson, and S. Floyd, "Wide-area traffic," in *Proceedings of the conference on Communications architectures, protocols and applications - SIGCOMM '94*, 1994, vol. 24, no. 4, pp. 257–268.

[6] V. Paxson, "Empirically Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Trans. Netw.*, vol. 2, no. 4, pp. 316–336, 1994.

[7] G. Gómez Sena and P. Belzarena, "Early Traffic Classification using Support Vector Machines," *Proc. LANC*, p. 60, 2009.

[8] W. Li, M. Canini, A. W. Moore, and R. Bolla, "Efficient application identification and the temporal and spatial stability of classification schema," *Comput. Networks*, vol. 53, no. 6, pp. 790–809, Apr. 2009.

[9] S. H. Yeganeh, M. Eftekhar, Y. Ganjali, R. Keralapura, and A. Nucci, "CUTE: Traffic Classification Using TErms," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, 2012, pp. 1–9.

[10] G. Aceto, A. Dainotti, W. de Donato, and A. Pescape, "PortLoad: Taking the Best of Two Worlds in Traffic Classification," in *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, 2010, pp. 1–5.

[11] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-related Features," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*, 2016, pp. 407–414.

[12] D. Wang, L. Zhang, Zhenlong Yuan, Y. Xue, and Y. Dong, "Characterizing Application Behaviors for classifying P2P traffic," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, 2014, pp. 21–25.

[13] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, 2014, pp. 797–801.

[14] M. S. I. Mamun, A. A. Ghorbani, and N. Stakhanova, "An Entropy Based Encrypted Traffic Classifier," in *Information and Communications Security*, 2016, pp. 282–294.

[15] J. Sherry *et al.*, "BlindBox," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 213–226, Aug. 2015.

[16] V. Aghaei-Foroushani and A. N. Zincir-Heywood, "A Proxy Identifier Based on Patterns in Traffic Flows," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 118–125.

[17] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through Weibull analysis of

dwell time," in *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10*, 2010, p. 379.

[18] F. Haddadi and A. N. Zincir-Heywood, "Benchmarking the Effect of Flow Exporters and Protocol Filters on Botnet Traffic Classification," *IEEE Syst. J.*, vol. 10, no. 4, pp. 1390–1401, Dec. 2016.

[19] S. Stibler, N. Brownlee, and G. Ruth, "RTFM: New Attributes for Traffic Flow Measurement." pp. 1–18, 1999.

[20] D. Arndt, "NetMate-flowcalc « Daniel Arndt," 2011. [Online]. Available: https://dan.arndt.ca/projects/netmate-flowcalc/. [Accessed: 04-Oct-2017].

[21] E. Frank, M. A. Hall, and I. H. Witten, "The WEKA Workbench Online Appendix for 'Data Mining: Practical Machine Learning Tools and Techniques' Morgan Kaufmann, Fourth Edition, 2016," *Morgan Kaufmann, Fourth Ed.*, 2016.